Community Place: Architecture and Performance

Rodger Lea Yasuaki Honda Kouichi Matsuda Satoru Matsuda
Sony Architecture Labs. Software Lab
Sony Corp. Shinagawa, Tokyo 146 Japan
Tel: +81 3 5448 4380 email rodger@csl.sony.co.jp

ABSTRACT

Community Place is a shared multi-user VRML system designed to work in the Internet. It consists of a VRML2.0 browser, a multi-user server architecture and an application support environment. Community Place has been developed over a two year period during which we have contributed some of our work to the VRML2.0 standard. The VRML community is now embarking on the next phase of VRML's development, support for multi-user systems. This paper discusses the architecture and performance from the product version of Community Place and our plans for future development. Its goal is to provide input and data to the debate about multi-user VRML in the hope of aiding the next phase of VRML's evolution.

Categories and Subject Descriptors: C.2 [**Computer - Communication Networks**]: Distributed Systems; C.4 [**Computer Systems Organization**]: Performance of Systems; I.3.7 [**Three-Dimensional Graphics and Realism**]: Virtual Reality

1 INTRODUCTION

The Virtual Society (VS) project is a long term research initiative that is investigating how the future electronic society will evolve. As a first step in this project we have been exploring the capabilities of existing technology to support social spaces, i.e. electronic locales where people go to interact. In our initial investigation, we have chosen to explore the 3D spatial metaphor as a basis for a shared information and interaction space. Our choice of a 3D spatial metaphor is based on our believe that such a metaphor is an attractive 'natural' environment within which users can interact. Rather than strive to find new metaphors to present data, we mimic the world in which we live. While it is clear that not all interaction needs or benefits from a three dimensional setting, we believe that such a setting, providing support for notions such as presence, location, identity and activity[1], will provide a generic basis on which a number of different application types will be constructed.

Thus, our goal has been to build a support infrastructure that will allow many users to participate in a shared, interactive 3D world. Such interaction will include the ability to see each other, talk to each other, visit locales with each other and work with each other. Our system, CommunityPlace[1] (CP) has elements of a computer-supported cooperative work (CSCW) environment, a virtual reality system and an on-line chat forum.

[1]Community Place was formally know as CyberPassage

Such systems have already been explored in a number of experimental research platforms. However in the majority of cases the work has been confined to high bandwidth communication networks supporting small numbers of users. Our work differs in that our initial goal has been large-scale systems capable of supporting many geographically dispersed users, interconnected through low bandwidth, high latency communication links.

This paper is laid out as follows; section 2 we briefly introduce the architectural possibilities when building a distributed Virtual Environment (VE) and present the issue of distributed consistency which any distributed VE must solve. In section 3 we introduce the basic CP system architecture, relate it to the architectures discussed previously and present each of the major components. In particular we introduce the two models we provide to build distributed shared behaviors, the simple shared script (SSS) model and the application object (AO) model. In section 4 we introduce our initial approach to server scalability based on the spatial model and section 5 concludes with an overview of the architectural features of the current CP system.

Section 6 then discusses performance issues relating to raw server performance, and communication costs for the two application models under different server loads. In section 7 we introduce our ongoing development of a replicated server to migrate the system towards a hybrid client-server, peer-to-peer system. Section 8 relates our work to others both in the academic and internet communities, and sections 9 and 10 discuss future directions and conclude.

2 ARCHITECTURES FOR SHARED VRML WORLDS

Building a simple distributed Virtual Environment (VE) is not difficult. It requires 3 conceptual components; a database of objects that exist in the world, a set of tools to populate that database and a set of devices that display the contents of the database. The display device doubles as an input device and allows users to navigate through the world and to interact with other users and objects in the world. To achieve this, it requires some form of communication that will allow the display devices to access the database and to propagate user input to the database.

- The display device can range from a low-cost consumer electronics device up to a high-end graphics workstation.
- The communications link is of prime importance to the performance of the user device. In a consumer setting, current technology constrains us to a maximum bit rate of 14k bits per second, whereas a modern research lab has access to a Gbit communication link.
- The server maintains the database of scenery objects that make up the world and users who are navigating through those scenes. It delivers the contents of the database to the display devices as and when needed.

There are two significant issues to be addressed when building a distributed VE. The first issue is the physical model used to structure the system, i.e. where the components go. The second issue is how those components are used to support the distributed algorithms, and in particular the consistency guarantees.

Since the scene database is shared by all client devices that are accessing the database, and since the clients will be updating the data, then the principal role of the database is to maintain a consistent copy of the data. Changes originating at the client side need to be propagated to the database, and used to update the scene in the database in a consistent manner.

2.1 Possible system architecture

Distributed systems are by their very nature large and complex systems, whose design is often more of an art than an engineering discipline. In this respect, distributed VEs do not differ from any other type of distributed application.

There exists a spectrum of choices when trying to decide how to build a distributed VE. This spectrum ranges from the traditional client-server architecture, through clients connected to replicated servers to fully replicated peer-to-peer systems.

- client-server architectures: The client-server architecture is conceptually and practically the simplest method of building a distributed system. The majority of small scale distributed systems are built in this way. The database of scene objects is held on a single server. The client devices access the server to obtain information about the world's structure, and to inform the server about changes originating at their client. Because there is only one copy of the scene data held at the server, then the issue of updating the data in a consistent manner is simple.

  This model is complicated slightly if client side devices, ie the browsers cache data. In this case the system has to be able to resolve the issue of inconsistencies between the copies cached in the clients. Generally, the system will arrange to cache read only copies at client, any updates will be made to the master copy held in the server and the master copy will update cached copies or simply invalidate them.
- peer-to-peer systems: At the other extreme from the client-server model, is the peer-to-peer model. In this architecture, there is no single repository or master copy of the data. Instead, each client maintains a copy or replica of the data. The current state of the world database is distributed throughout the browsers.

  Updates to local copies of a replica have to be made to all replicas. Again, there are a many protocols, all of which aim to achieve the same result; ensuring that a change to one copy is replicated to all other copies to maintain consistency.
- Hybrid systems: Somewhere in between these two endpoints on the spectrum are the hybrid systems. These systems merge the client-server and the peer to peer model. They generally maintain the client-server link from the browser to the database. However, they replicate the database for performance reasons.

  By replicating the database, the system is able to avoid the critical drawback of the client-server model, that of performance bottleneck and single point of failure due to the single server that all communication goes through.

2.2 Consistency architecture

The fundamental model presented by a distributed virtual environment (VE) platform is one of a shared 3D space. Such a space, because it is shared, must be seen consistently by all users of that space. A system can provide different levels of consistency, ranging from a strict interpretation to best effort[13].

In a strict interpretation, any actions that occur in the shared space must be propagated to all participants in that space, and conflicts between user actions are either avoided, or resolved. Furthermore, actions in the space maintain their causal relationship so that a user can make sense of a 'happened before' and 'happens after' relationship. Obviously, maintaining such consistency in a system where there are many participants is a complicated task and one that requires significant exchange of information between the copies. The choice of algorithm is crucial to the amount of message passing needed to reach consistency. Any distributed consistency algorithm has two major concerns:

- Membership: The membership of the consistency group, i.e, who is taking part in the consistency algorithm is crucial to performance. Any mechanism that reduces the number of participants in the consistency group directly reduces the number of messages that must be exchanged.
- Consistency guarantee: Once membership has been decided, the next issue is what model of consistency is used by the consistency algorithms. There has been much work in the research community addressing the issue of distributed consistency in more traditional data applications with a goal of reducing the cost of the algorithms. This work has concentrated on relaxing the degree of consistency either in a temporal domain[21] [22], or in a data value domain[12].

It is interesting to note that the design of the consistency algorithm, while related to the underlying system architecture is, to a certain extent, an orthogonal issue. For example, it is possible to have a fully replicated, peer-to-peer system model yet still use a single master algorithm for data consistency. Alternatively, it is possible to use a single server, yet to run consistency algorithms between cached copies in client browsers.

3 CP SYSTEM ARCHITECTURE

The Community Place system is a client-server system. It consists of a central server, running at a well know internet node, which is responsible for enabling browsers to view a single shared VRML scene. Browsers connect to this server when they load the VRML file associated with the shared scene.

The basic system architecture for CP is shown in figure 1. In the following sections we discuss the individual components in detail.

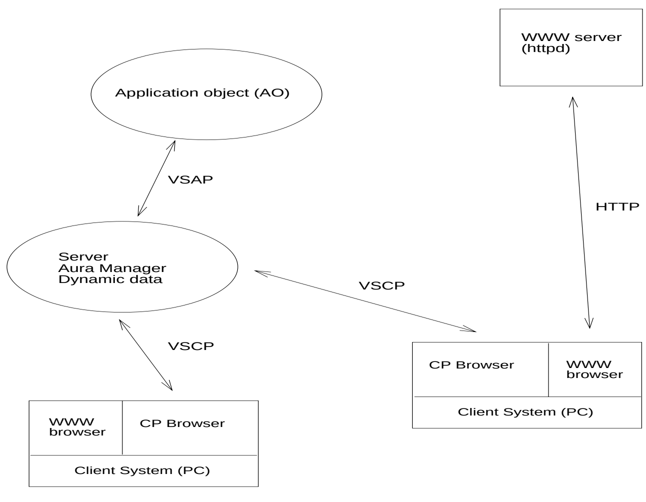

Figure 1: CP architecture

3.1 Browser

As can be seen from figure 1, the browser works in conjunction with a HTML browser. The CP browser loads the 3D data file (in VRML2.0 format), in the course of which it finds an entry describing the location of the server to be used for this shared 3D scene. The CP browser then contacts the server via the Virtual Society Client Protocol (VSCP) that runs above IP. The server informs the CP browser of any other users in the scene, including their location, and any other 3D objects not contained in the original scene description downloaded from the web server. The details are discussed below.

Local scripting

The CP browser supports the VRML2.0 standard and uses Java as its scripting language. In the usage scenario discussed above, a VRML file is downloaded to the local browser which renders its contents. CP uses the associated HTML browser to subsequently download any scripts referred to in the VRML file. Scripts are able to manipulate scene graph nodes by generating events that are delivered to the node and change one or more of its properties, for example, its position in the scene, its shape or one of its material attributes. Obviously, since the scripts are fully functional Java code, they are not restricted to just changing the scene graph. They can, for example, dynamically generate additional VRML nodes, or locate and add existing VRML to the base scene downloaded in the original VRML file. This may be carried out using a call to a http server or by a request to another network machine. Further, they can also interact with other applications, for example mining data from a database which can subsequently be turned into VRML and added to the shared scene.

In a standalone browser, the execution mechanism of sensors, events and scripts allows animation of a local scene graph. However, to support our goal of shared interactive scenes we allow scripts to communicate events to the scene graphs managed by other browsers.

Browser-server communications

The browser communicates with other browsers using the server (see below) and a protocol called Virtual Society Communications Protocol (VSCP). VSCP has two goals: efficient communication of 3D scene transformations and open-ended support for script specific messages.

The first goal is answered by ensuring that VSCP has a very compact representation of 3D transformations and control messages. This efficiency is obviously crucial considering our target of dial-up connections. For the second goal, VSCP has an object-oriented packet definition that allows applications to extend the basic packet format with application specific messages. VSCP runs above TCP which provides connection guarantees and simple firewall traversal.

This mechanism enables us to send and receive script level messages that allow the browsers to share events and so support shared interaction with the 3D scene. For example, a local user event causes a local script to run, which in turn uses the message sending facility of the CP system to deliver the event to a remote browser sharing the scene. At the remote browser, this network event is transformed into a local event which in turn causes execution of the local script. We discuss this mechanism in more detail in section 3.3.

3.2 Server

The server, known as the CP Bureau acts as a position tracker and message forwarder. Each user's browser, as it navigates through the shared scene, sends position information to the server. The server then uses AOI (area of interest) algorithms (see section 4.1) to decide which other browsers need to be aware of these position changes. The server sends out the position to the chosen browsers, which in turn use the information to update the position of the local representative, the avatar, of the remote user. The role of the server is limited to managing state on behalf of connected users. It is generally unaware of the original scene loaded by the browser.

The second role of the server is to carry out a similar function for any script level messages that are generated by a browser as a result of user interaction. Again in a typical scenario, a user event, such as a mouse click, will cause a local script to run. This script will update the local scene graph and then post the event (or the resulting change) to the server. The server then re-distributes this message to other users in the scene so that the scene update is replicated and shared by all users. We refer to this approach to application development as simple shared scripts (SSS).

3.3 Application programming models

The CP system provides two models for application building, the first is known as the Simple Shared Script (SSS) model, and the second as the Application object (AO) model. The two share some elements but are targeted at different applications and different authors. Both models use the message sending API that CP supports. Messages can be sent to all browsers (SENDTOALL), to all except the sender (ALLNOTSELF) and to the owner or master of an object (RESPONDER).

Simple shared scripts

The SSS model is a simple mechanism designed for small shared applications in the 3D world. The model is a replicated script model with each browser downloading the same script and executing it locally. Typically these scripts would be associated with objects that are downloaded in the initial VRML file.

As discussed above, the VSCP protocol supports script message sending allowing a local script to send a message to all other browsers sharing the scene. Using this mechanism, it is possible for scene authors to develop small scale applications that share events by sending those events to other browsers via the server.

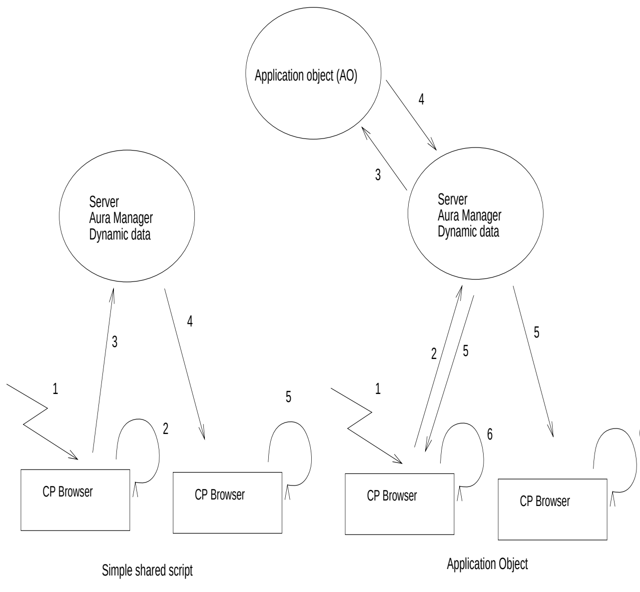

Figure 2: SSS versus AO scripting

In figure 2 we can see message flows as a result of a user selection (i.e. a mouse click) in the SSS model (left side). A user selection (1) causes a local script to run (2). This in turn converts the event into a message and sends it to the server (3). The server sends the message to all other browsers (4) who then convert the message to an event that causes execution of the local script (5).

To deal with the issues of ownership and persistence, we add a notion of a master browser to the SSS model. The master browser is selected by the server and told it is master. Scripts are able to send events either to all other browsers, or to the master. In situations where, for example, the user wants to implement serialization of a scene object, all interactions with that object use messages that are sent to the master browser (RESPONDER). The master browser then makes changes to its local copy, and distributes those changes by using the SENDTOALL form of the message send.

Application objects

While the SSS approach is suitable for a number of simple shared scene updates, more complicated applications require a more sophisticated mechanism. To support this, CP has a notion of an application object (AO) which exists externally to the browser and the server. The application object is an application run time that allows application builders to create 3D objects, with associated behaviors, and to inject them into existing shared scenes. It allows users, via local scripts, to interact with these applications. The applications use the Virtual Society Application Protocol (VSAP) to register their application objects with the server. Registration informs the server about the 3D visual representation, written in VRML, and the spatial positions of the objects. The server then informs the relevant browsers about the existence of these application objects and the VRML file to be downloaded to display them. Lastly, the server forwards application-specific messages between the AO and the browsers. Thus, an AO consists of three parts: the 3D data description that represents the application in the shared scene; the associated scripts that accept user input and communicate back to the AO; and the AO side code that implements the application logic.

The application model presented by the AO is subtly different from the SSS model described above. In particular, the AO defines, by default, a master or controller for the application, whereas in the SSS model, the scripts are essentially peer-to-peer and can make use of the master concept if required. More importantly, the AO mechanism, because it registers objects via the server, benefits from the server's use of Area of Interest (AOI) algorithms to reduce communications (see 4.1). In the SSS model, scene objects downloaded in the original VRML file are not known to the server and so it is unable to optimise message sending.

Returning to figure 2 in the AO model (right side), the user event (1) causes a message to be sent to the server (2), which in turn sends the event to the AO managing the selected object (3). The AO carries out internal processing and then typically sends back a message (4) via the server to each browser (5) that runs the local script (6). There are obviously many variations within these models. However the major difference is that in the AO model, there is a designated owner for an object who has sole control over its update.

A key aspect of the AO model is that it allows dynamic addition of VRML data and associated scripts to an existing scene. The feature allows us to build shared worlds that evolve over time. The basic scene description is set up in a base VRML file and downloaded by browsers. Subsequently, new scene elements can be added by creating AOs to manage the new elements, and by using the server and the VSAP protocol to add the new scene element to the basic model already loaded by browsers. In a commercial environment, this allows service providers to dynamically inject an application into an existing shared scene. For example, a 3D shopping mall would consist of a basic 3D scene which is downloaded initially by the user. Subsequently, service providers can add shops into the scene by creating AOs and connecting to the server. This model allows a decoupling between server managers and service providers, thus providing an open and extensible mechanism for application provision.

4 SCALEABILITY

In the previous section, we discussed the basic architecture of the CP system and the main components. To allow this architecture to scale we exploit the following aspects:

- Static scene data is downloaded initially as part of the VRML file and replicated at all browsers. Dynamic data can be managed using local scripts plus message passing. This reduces the burden on the server because it does not need to manage this scene data.
- We offload some processing into the client browser using the local scripting facility. This allows us to send events, rather than state changes, and to use local scripts to handle the events. This enables such techniques as dead reckoning[10].
- Sophisticated applications can be managed by external processes and can use the local script to manage local updates in individual browsers. Again, this approach reduces the role of the server to a message forwarder and the management of the application data is split between the AO's and the browsers.

Although these mechanism do allow some degree of scaling by reducing the communications between browsers (via the server), they are not sufficient to support our goal of many hundreds of users interacting in a shared space. To achieve such scalability, it is necessary for us to find a way to limit the number of messages needed between browsers to support the shared scene.

4.1 Spatial areas of interest

In previous experiments with the Dive system [9], we have observed that participants form sub-groups where activities occur in clusters or peer-to-peer within the global session. This mimics the way we use the spatial model in the real world. The observation can be exploited to decrease overall message passing if one can deliver packets only to the recipients they are intended for, i.e. those within the sub-group. In this way, the amount of global traffic is limited, and the number of incoming messages to each user is reduced.

Using the three dimensions of space is a well-known approach to partition VEs into several disjoint AOIs. Static geographical regions are used in applications based on natural terrains, such as in DIS based systems[16].

A different approach uses intersecting volumes to model interaction between participants. This notion of a spatial area of interest associated with a user has evolved out of work in the COMIC project [3]. The spatial area, known as an *aura* determines a boundary; objects or users outside the boundary can not be influenced or interacted with. In contrast, all objects within the boundary are candidates for influence or interaction. The COMIC model goes further by defining two notions, *focus* and *nimbus*, to represent the degree of interest users have in each other. The focus represents the degree of interest one user brings to bear on another. The nimbus represents the degree of attention one user pays to another. The combination of the focus and nimbus of two interacting users defines their level or degree of interaction.

It is this model that we seek to use to drive our consistency mechanism and to reduce the number of participants in any consistency algorithm.

To achieve this, the server is structured as shown in figure 3. An aura manager is responsible for tracking the spatial location of any user (or AO object) and for determining if two user's auras have collided. If they have, the aura manager causes those two objects to join a consistency group which is defined as a set of objects who

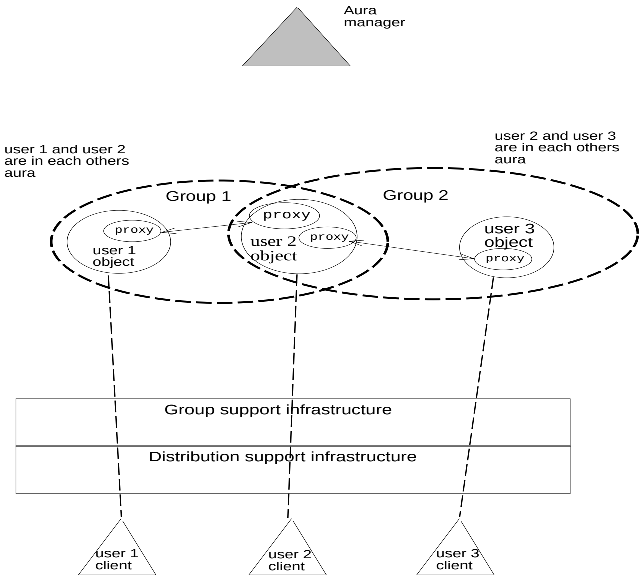

Figure 3: Auras and groups

have shared data which must be maintained consistent. For example, in figure 3, user 1 and user 2 are in each other's aura, user 2 and user 3 are in each other's aura, but user 3 is not in user 1's aura. Thus, any updates to user 3, e.g. a position update, will be sent to user 2 but not user 1. The actual replicas are denoted by proxies, i.e. local representatives of the remote object. In the case where the objects are all local to one server, these proxies are generally pointers to the master object.

In essence, the aura manager is responsible for defining groups of spatially co-located objects who need to maintain a degree of consistency. As it decreases the degree of sharing, this mechanism is used to reduce the amount of information that has to be sent out from the server as a result of any state changes.

5 COMMUNITY PLACE ARCHITECTURE: CONCLUSION

Although the architecture of the CP system is technically client-server, because of the way the server is structured, it also has elements of a peer-to-peer architecture.

The server does not maintain a database of the VRML scene. Rather, data is replicated at each browser. If the server acted simply as a message redistributer, the system would actually be a peer-to-peer architecture. However, the CP Bureau acts both as a message replicator and a shared database. Since it is responsible for maintaining the location and some attribute data belonging to clients (i.e. their avatar data), then the total state of the shared VRML scene is split between VRML data managed at the clients and data managed by the server.

This approach has been adopted simply for performance and scaling reasons. Our goal is to support large-scale worlds, with thousands of participants. To achieve that, we have tried, where possible, to minimize the data held in the server, since the server acts as a bottleneck in the system.

By pushing data out to the clients, we are able to both reduce server load and increase performance, because the client has local copies of data and is therefore not forced to access the server when it needs new data.

The downside of this is that certain data, in our case data belonging to VRML entities that have behaviors associated with them, is replicated at clients and we need to run consistency algorithms between client browsers.

Again, the approach taken in CP is a hybrid one. The SSS mechanism supports both fully replicated scripts and master slave scripts and allows scene authors to decide which to use.

The AO model implements, by default, a master model with a single copy of the data. However, application authors are free to cache data at the client and implement their own consistency algorithms.

6 PERFORMANCE

Because the CP system uses a range of techniques for performance and provides a set of mechanism for application builders, there is no simple performance data that accurately capture the performance of the system. To allow readers to gain some understanding of the performance of the system under different usage conditions, our experiments are divided into two areas: the performance of the multi-user server as a result of its AOI algorithms, and the performance of the browser-to-browser communications when using the two application models we offer.

6.1 Server performance

In the first set of experiments, we load the server with an increasing number of clients. Performance is measured for three machines. A Sun Sparc UltraServer 170 running Solaris 2.5 with 320MB of memory, an SGI Indigo2 running Irix 5.3, with 256MB of memory and a Sony NEWS 5000 machine running NEWSOS6.1 and 256MB of memory.

The client side machines are Sony NEWS workstations. Each of them supporting a client side test harness and connected to the server machine using a 10Mbit Ethernet. The Ethernet is at all times lightly loaded with general day to day network traffic.

The client side test harness is a fully configurable program that can be used to simulate actual browser usage. The configuration parameters fall into four categories:

- Client numbers. The harness can be configured with any number of clients, each will generate a separate connection to the server. For all client, the percentage who move per second can also be specified.
- Chat messages. For each client the size and frequency per second of text messages can be specified.
- Application message. For each client, the size and number of application messages per client can be specified.
- Attributes. For each client, the size of the client specific attributes, managed by the server, can be specified.

For each client managed by the test harness, the client connects to the server, exchanges some set up messages and then stores some attribute information at the server. The client then begins to move, send text and send application messages according to the test harnesses configuration parameters.

The amount of processing and messages generated as a result of any one client moving depends on the configuration of the server with respect its area of interest (aura). The area of interest algorithms use two crucial parameters, size of area of interest (AURASIZE) and maximum number of avatars in an aura (MAXINAURA). The first parameter dictates the size of the bounding box for avatar aura collision calculations at the server. The second dictates how many of the potential collisions are reported to the client. If the aura size is large, but the maximum number of avatars in an aura is restricted to a small number, then a client will only ever be informed of a small subset of potential collisions, where the subset equals the maximum number of clients allowed in an aura. Messages sent by a client are replicated to all browsers who have that client in their aura. Hence, the MAXINAURA is a rough indication of the replication factor for each message.

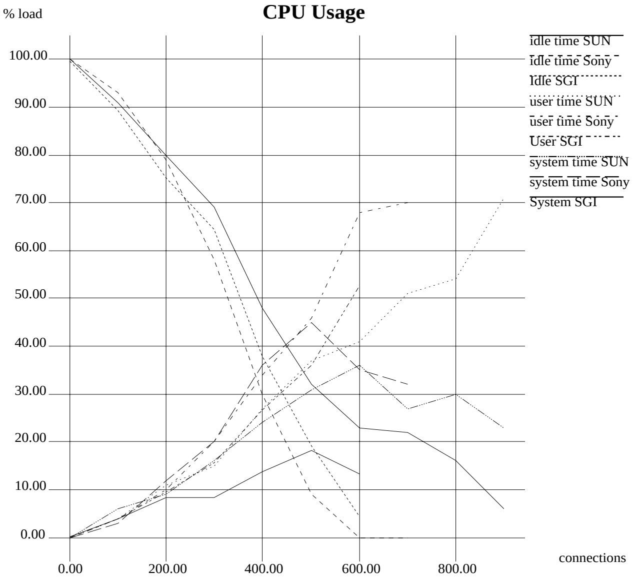

Figure 4: CPU load at the server

As can be seen from figure 4, on a Sony NEWS workstation, assuming a reserve of 30% CPU idle time, then the server can support 400 connections. The SGI machines, at the same load, will handle approximately 425 connections and the Sun 170, approximately 520 connections. The figures correspond roughly to the relative performance of the 3 machines, the Sony and SGI are comparable, and the Sun 10% more powerful. This implies that a larger server machine would support a correspondingly larger number of connections.

However, the network traffic generated by the server is also an important consideration. Figure 5 shows this information.

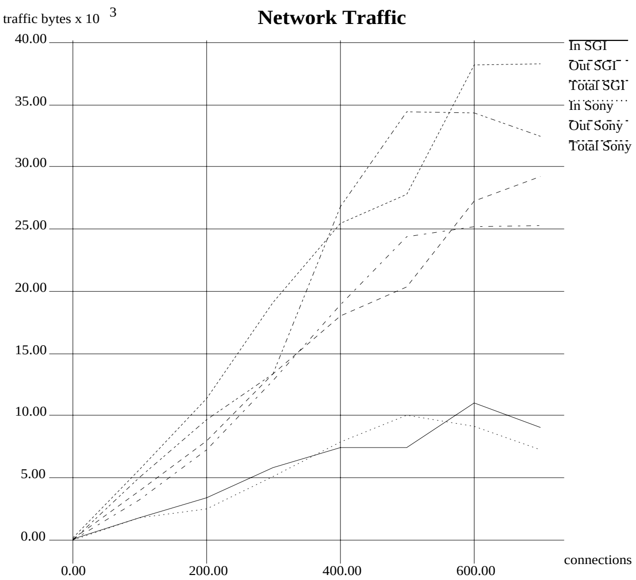

Figure 5: Network traffic at the server

The network traffic indicates the amount of data arriving at the server from clients, and the amount of data leaving the server heading for clients. Generally the data leaving the server will be greater than that which arrives, because the server is replicating the data and distributing it to various clients. As can be seen, the Sony NEWS will support a maximum of 500 connections, before network performance tails off, the SGI's performance is similar. Although not shown, the performance of the Sun machine is correspondingly better and supports approximately 630 connections.

At these points, all servers are handling approximately 350k bytes of network traffic. The reader should note, that for the purposes of these experiments, the configuration parameters are set to cause an artificially high amount of message traffic. Using more realistic figures collected from actual usage at our public servers, a Sun workstation will support upto 1000 connections.

As discussed in section 6.1 the area of interest algorithms use two parameters to reduce network traffic; AURASIZE and MAXINAURA. To understand the effect of varying these parameters we present two further experiments.

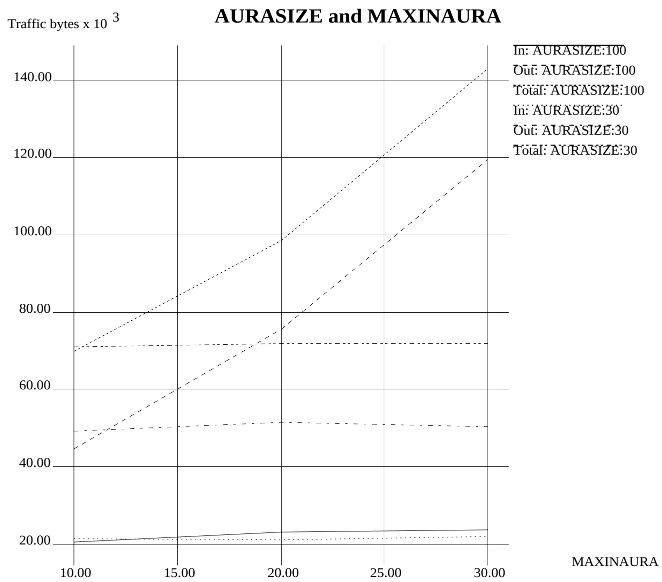

Figure 6: Server traffic as a function of MAXINAURA and AURASIZE

In the first, the aura size is set to a higher value of 100 meters and the MAXINAURA value gradually increased. The network traffic is measured and presented. In the second experiment, we keep the AURASIZE at 30 meters, and alter the MAXINAURA. In both cases the number of connected clients is held constant at 100.

Looking at figure 6 the total message traffic using an AURASIZE of 30 meters stays constant as the possible number of avatars in an aura increases.

In contrast, if the AURASIZE is increased to 100 meters, the effect of allowing more avatars in an aura is a marked increase in total message traffic. Looking in more detail, although the traffic from clients into the server remains constant (at about 200k) the traffic generated by the server starts at 420k when MAXINAURA is 10, and climbs to 1400k when MAXINAURA is 30.

These experiments show clearly the difficulty in predicting message traffic (and hence server load) based on the AOI algorithms. While it is clear that increasing the MAXINAURA will lead to a growth in traffic, the interplay with the AURASIZE is less clear. The actual result of these two factors is dependent on the movement behavior of the clients which in turn is based on the scene design.

6.2 Browser performance

We conducted five experiments to show the basic performance of the browser and server interaction in the CP system. In all cases the server machine is a NEWS 5000 and the client machines are DEC Celebris PCs (Pentium 133MHz 64MB - Win95).

The first experiment measures the event processing cost within the browser. We measured the cost of calling Java from VRML (the cost of eventIn) and posting an event from Java to VRML (the cost of eventOut). The actual experiment is as follows: we have two script nodes in a VRML file, each of which refers to a different Java class file. On an initial trigger event, one of the script nodes starts to generate events in the Java code. These events are routed to the other script node. The execution path is: Java → eventOut → routing → eventIn → Java. The destination Java script code performs nothing. The cost of this execution path is *0.99ms*. (averaged over 3000 runs).

The second experiment measures the cost of basic networking function. It measures the cost of sending a message from a Java program in a browser to a server which sends back the message to the same client. The actual setting is as follows: a browser loads a simple VRML world and connects to the server. On the initial trigger event in the world, some Java code is called and the time measurement started. This code sends a message (using our own API) to the server. The server automatically sends back the message to the browser. On receiving the message, the browser generates an event within the VRML scene which is routed to the Java code. The execution path is: Java → messaging API → network → server → network → eventIn → Java. This code path is repeated 3000 times for an average cost of *6.48ms*.

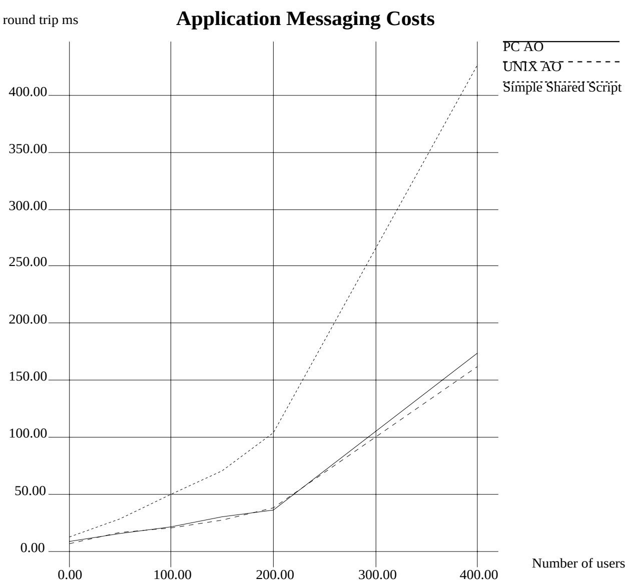

Figure 7: Total system communication costs

The other three experiments show the different costs for different application models in a practical setting. As described in 3.3, we have two different application programming models, *Simple Shared Scripts* and *Application Object*. In both cases, we create a simple VRML application that generates an initial event in the browser which is then delivered to a master (in case of SSS model) or AO (in case of AO model). On receiving the message, the master or AO sends the same message to all the clients (including the client that generated the original event). The execution path looks like: Java → messaging API → network → server → network → AO or master → network → server → network → eventIn → Java. This test is repeated several hundred times and an average value taken.

For each of these execution paths, we load the server with an increasing number of test clients and measure the effect on the total round trip time.

Currently we provide AOs on a PC platform where applications are written in Java and on a UNIX platform where C++ is used. For comparison, we used both PC AO and UNIX AO for the measurement. Figure 7 shows three graph lines: one for SSS and the other two for PC AO and UNIX AO.

The graph shows clearly that the SSS model is both more costly, and more sensative to server load. The reason for this is that the SSS model does not use the servers internal AOI algorithms. Thus, when the master replies via the server, the server is forced to send the reply to all connected clients.

In contrast, the AO model uses the AOI optimisation available in the server. Thus, the maximum number of messages generated when the reply is recieved from the AO is dependent on the MAXINAURA parameter of the server. In these experiments it is set at 8. The PC AO and Unix AO performs in a similar manner, which is expected since their task is to simply receive the message and immediately send it back to the server.

7 DISTRIBUTED ARCHITECTURE

Our current development work is designed to scale the client-server architecture beyond its present limits. To attack this issue we are investigating a hybrid client-server, peer-to-peer model with our colleagues at the Swedish Institute of Computer Science.

As part of this work we are building a replicated version of our current server. By replicating the server, we are able to spread the processing and communication load between several servers and so scale the entire system.

In this architecture, the aura manager tracks objects and informs them of any aura collisions. The replica joins the communication group associated with the remote object and runs the consistency algorithm defined for that object. However, in the non replicated server, the group was a structuring technique within the server, within the distributed server; the group maps to a multicast communication group.

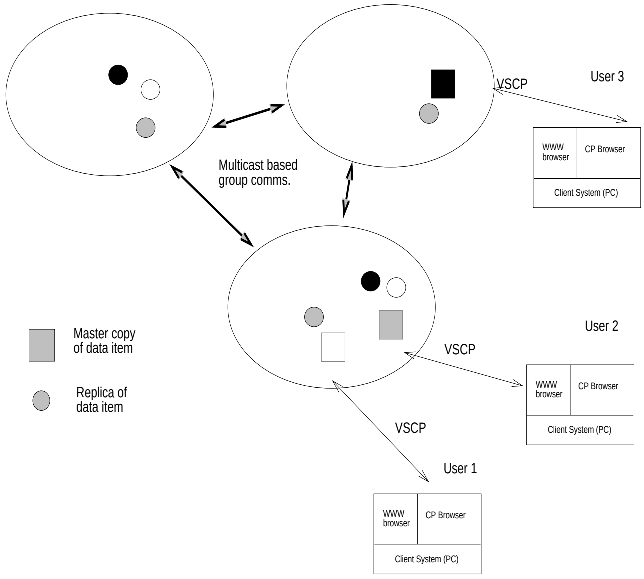

Figure 8: Distributed architecture

In the distributed server case, the spatial model is used exactly as in the single server case. It partitions the database into groups of spatially co-located objects who manage their consistency using a group communication model. This allows us to reduce the amount of data that must be replicated at each server to that which is required for the groups associated with users actually connected to that server.

Our communication mechanism is based on multicast which is used between servers to support the consistency algorithms needed.

Multicast communication allows a single message send to be delivered to a group of receivers. In hardware supported multicast environments, e.g. ethernet this allows for very efficient messaging. In the Internet, an experimental multicast layer built above IP is used. This system, known as the MBone, implements a virtual multicast network over the inherently point to point mechanism of the internet. The technique used is based on message encapsulation and tunneling.

Further details of the aura model and its use of the group communication model can be found in [8] which reports on joint work between our group and the Dive group at SICS.

Our initial investigations of using this multicast group mechanism to support weak and adaptive consistency models is based on previous work with Apertos[6].

8 RELATED WORK

There is considerable research activity in the area of large scale distributed environments, these include projects focusing on collaboration [9], [19] [11] [5] [14] [15] and projects focusing on simulation [20] [17]. In most of this work, the emphasis has been on workstation level devices and high bandwidth communications.

As discussed in the text, the Dive system was the original testbed for most of the work on the spatial model we used. MASSIVE inherited this spatial model and carried out a fuller implementation. Our implementation of the spatial model, particularly the use of the aura collision manager is based on both Dive and MASSIVE. However, the main use of the aura model in the CP system is to reduce communications to enable scaling, our use of multicast communications supports this. Our work also differs in the target. Both MASSIVE, and to a lesser extent, Dive, have concentrated on collaboration and conferencing and have assumed professional level computer and communication facilities. Our main goal has been large-scale social worlds using low-cost consumer equipment. As such, the eventual architecture we adopted, a hybrid client-server/peer-to -peer model differs from the more 'pure' approaches of Dive and MASSIVE.

In terms of social shared spaces targeted at the consumer market, there is already a legacy with systems such as habitat[23] and Worlds Away - a Compuserve service based on Habitat, which, although not full 3D spaces, offer some degree of spatial metaphor. More sophisticated shared spaces have been built by Worlds Inc., including the Worlds Chat and the Alpha world[2]. However, these projects, although using the Internet, have relied on proprietary graphics and browsers.

Recent work targeting the WWW and using full 3D shared spaces has mainly been confined to the VRML community. Within that community there are several projects of note. The Cybergate system from Blacksun Inc.[3] has built and experimented with shared 3D spaces similar to the CP project. However CyberGate concentrates on multi-user but does not support shared behaviors. Moondo[4] is a similar system to CyberGate in that it currently supports only static scenes. However, Moondo has experimented with a shared object model as a basis for shared consistent objects.

The Pueblo project from Chaco Communications[5] has evolved out of earlier work on social MUDs. Recently it has augmented the MUD server with VRML support and provided a VRML1.0 browser that allows MUD authors to build 3D scenes. This approach allows world builders access to the rich mechanism of the MUD database, but again only supports static scenes.

[2] http://www.worlds.net

[3] http://www.blacksun.com

[4] http://www.intel.com/iaweb/moondo/index.htm

[5] http://www.chaco.com

9 CURRENT AND FUTURE DIRECTIONS

We are currently working in three broad areas. Firstly, we are extending the media support within the shared scenes. We have therefore experimented with audio chat facilities. However, the low-bandwidth links to home PCs severely curtail the fidelity of the audio stream. We are also designing streaming mechanisms for various media to allow us to stream audio and video from AOs, via the server, to the browser. Again the principal constraint is bandwidth.

Our second area of interest is augmenting the application mechanism with application libraries that allow simpler creation of complicated applications. To date, authoring consists of using a 3D modeler to build the basic components. An authoring tool, called CP conductor, is then used to assemble objects into the scene, and to subsequently associate behaviors with those objects. The authoring tool provides a set of pre-defined scripts that can be dragged and dropped onto objects in the scene, allowing easy development of simple scenes. However, more complicated behaviors have to be written by the scene author. We aim to provide a set of more sophisticated objects and associated behaviors, and to enable users to move these objects between independent scenes. A particular area of concern is inter-object interactions. This basic facility will allow a far richer space as it will enable users to claim ownership of objects.

Lastly, we are continuing our work on scaling issues in order to support larger numbers of users and more complicated scenes. As discussed in the text, part of this work is concentrating on the issues of consistency in large scale VEs, where we are exploring adaptive techniques to deal with the wide area communication problems.

10 CONCLUSION

One of the major reasons for the success of the WWW is that it has enabled unsophisticated users to participate, both as consumers and, more importantly, producers of information.

However, the WWW remains an essentially 'lonely place'. Although many users may be simultaneously viewing the same information, there is no support to allow them to interact, or even be aware of others.

Our goal has been to enable interactions, so that the WWW moves from being an information space to being a social space. To do that, we have chosen to use the 3D spatial metaphor to build 3D spaces that mimic real world spaces and provide a virtual place for interaction.

Although this is a necessary first step, it is not, we believe, sufficient to cause interaction to take place. It is our belief that this type of large scale social interaction will only happen if users can create spaces to reflect their requirements. As such, the most important goal of the CP project has been to provide an infrastructure that allows easy creation of such spaces, within a familiar framework, the WWW.

While the main focus of our work has been on social spaces rather than on spaces that support more traditional CSCW tasks, we have built simple examples of worlds where collaboration is possible and well used. It is our belief, that by providing a platform that is sufficiently rich to support collaboration, but sufficiently open and accessible to allow anybody to author spaces, we will enable far greater use of the Internet for collaboration.

11 ACKNOWLEDGEMENTS

We are indebted to our colleagues in the Sony Computer Science Lab. and Sony's Architecture Labs. for their help in the definition of this project. We also wish to thank our colleagues at the Swedish Institute of Computer Science who have contributed indirectly to the CP design as part of our joint research project, Wide area virtual environments (WAVE). Lastly, our thanks, as always, go to Mario

Tokoro, Toshi Doi and Akikazu Takeuchi for their continuing support.

References

[1] Benford. S, et al. Networked virtual reality and cooperative work. Presence. Vol. 4 No. 4. Winter 1995. pp. 364-386. MIT Press.

[2] The VRML2.0 specification. Version 2.0, Final Working Draft, ISO/IEC WD 14772 July 18, 1996 Currently only available as. http://vrml.sgi.com/moving-worlds/

[3] Benford, S., Fahlen, L., Greenhalge, C. and Bowers, J. Managing mutual awareness in collaborative virtual environments. Proc. ACM SIGCHI conference on Virtual reality and technology (VRST'94) August 23-26th 1994, Singapore, ACM Press.

[4] Honda, Y., Matsuda, K, Rekimoto, J and Lea, R. Virtual society. Procs. of VRML'95, San Diego. USA. Dec. ACM press 1995 pp. 109-116, Order no. 434953 Available at: http://www.csl.sony.co.jp/project/VS/VRML95.ps.Z

[5] Broll, W. and England, D. Bringing worlds together: adding multi-user support to VRML. Procs. of VRML'95, San Diego. USA. Dec. ACM press 1995 pp. 87-94, Order no. 434953

[6] Lea, R. and Yokote, Y. Adaptive operating system design using reflection. Procs. of the 5th Workshop on Hot Topics in Operating Systems (HTOS-V). Orcas Island Washington, USA.1995. pp. 95-101. IEEE press. Also available as: http://www.csl.sony.co.jp/person/rodger/htos.ps.Z

[7] Lea, R., Raverdy, P.G, Honda. Y, and Matsuda, K. Issues in the design of a large scale VE. Procs. of HICSS-30 Minitrack on distributed VEs. IEEE press. Hawaii, Jan.7-10 1997.

[8] Hagsand, O., Lea, R. and Stenius, M. Using spatial techniques to reduce message passing in a distributed VE. These proceedings. Also avaiable as: Sony Computer Science Lab. Tech Report http://www.csl.sony.co.jp/person/rodger.html

[9] Carlsson, C. and Hagsand, O. DIVE - A platform for multi user virtual environments. Computer and Graphics Vol. 17. No. 6 1993 pp. 663-669

[10] Singhal, S. and Cheriton, D. Exploiting position history for efficient remote rendering in networked virtual reality. Presence. Vol. 4 No. 2. Spring 1995. pp. 169-193. MIT Press.

[11] Snowdon, D. and West, A. AVIARY: Design issues for future large scale virtual environments. Presence. Vol. 3 No. 4 Fall 1994. pp. 288-308. MIT press.

[12] Pu, C. Relaxing the limitations of serializable transactions in distributed systems. Proceedings of the 5th ACM European Workshop, Le Mont St Michel, France. Operating Systems Review Vol. 27. No. 2 pp. 66-71. ACM press.

[13] Mosberger, D. Memory consistency models Operating Systems Review Vol. 27. No. 1 pp. 18-26. ACM press.

[14] Shaw, C., Green. M., Liang, J. and Sun, Y. Decoupled simulation in virtual reality with the MR toolkit. ACM trans. on Information Systems. 11(3), pp. 287-317.

[15] Bricken, W. and Coco, G. The VEOS project Presence. Vol. 3 No. 2. Spring 1994. pp. 111-129. MIT Press.

[16] DIS ANSI/IEEE std 1278-1993. Standard for information technology, Protocols for distributed interactive simulation. March 1993

[17] Pope, A, The SIMNET network and protocols. BBN report no. 7102. BBN systems and technologies, Cambridge, MA. USA. 1989.

[18] Singh, G., Serra, L., Png, W. and Ng H. Bricknet: A software toolkit for networked virtual worlds. Presence. Vol. 3 No. 1. Winter 1994. pp. 19-34. MIT Press.

[19] Greenhalge C. and Benford, S. MASSIVE: a distributed virtual reality system incorporating spatial trading. Procs of the 15th ICDCS. May 30 - June 2, Vancouver Canada 1995. IEEE press.

[20] Macedonia, M., Pratt, D. and Zyda, M. NPSNET: A network software architecture for large scale virtual environments. Presence. Vol. 3 No. 4. Fall 1994. pp. 265-287. MIT Press.

[21] Hutto, P and Ahamad. M. Slow memory: Weakening consistency to enhance concurrency in distributed shared memories. Procs. of the 10th ICDCS. pp. 302-311 May 1990 IEEE press.

[22] Birman, K., Schiper, A. and Stephenson, P. Lightweight causal and atomic group multicast. In ACM transactions on Computer Systems 9(3), 272-314. 1991

[23] Morningstart, C. and Farmer. F. The lessons of LucasFilm's Habitat In, CyberSpace: First steps. M. Benedikt. MIT press. Cambridge, Massachusetts, USA, 1992.